Discrete and Continuous Optimal Control for Energy Minimization in Real-Time Systems

Bruno Gaujal
Univ. Grenoble Alpes, Inria,
CNRS, Grenoble INP,
LIG, 38000 Grenoble, France.
bruno.gaujal@inria.fr

Alain Girault
Univ. Grenoble Alpes, Inria,
CNRS, Grenoble INP,
LIG, 38000 Grenoble, France.
alain.girault@inria.fr

Stéphan Plassart
Univ. Grenoble Alpes, Inria,
CNRS, Grenoble INP,
LIG, 38000 Grenoble, France.
stephan.plassart@inria.fr

***Abstract*—This paper presents a discrete time Markov Decision Process (MDP) to compute the optimal speed scaling policy to minimize the energy consumption of a single processor executing a finite set of jobs with real-time constraints. We further show that the optimal solution is the same when speed change decisions are taken at arrival times of the jobs as well as when decisions are taken in continuous time.**

***Index Terms*—Optimal Control, Real-Time Systems, Markov Decision Process, Dynamic Voltage and Frequency Scaling.**

I. Introduction

Minimizing the energy consumption of embedded system with real time constraints is becoming more and more important. This is due to the fact that more functionalities and better performances are expected from such systems, together with a need to limit the energy consumption, mainly because batteries are becoming the standard power supplies.

The starting point of this work is the seminal paper of [7] that solves the following problem: Let $(r_i, c_i, d_i)_{i\leq K}$ be a set of K jobs, where r_i is the *release or arrival* date of job i, c_i is its *size* (number of operations needed to complete the job) and d_i is its relative *deadline*. The problem is to choose the speed of the processor as a function of time $s(t)$ such that the processor can execute all jobs before their deadlines, and such that the total energy consumption J is minimized. In our problem, J is the *dynamic* energy consumed by the processor: $J = \int_0^T Q(s(t))dt$, where T is the time horizon of the problem and $Q(s)$ is the power consumption when the speed s is used.

This problem has been solved in [7] in the *off-line* case, *i.e.*, when *all* jobs are known in advance, and when the power function $Q(\cdot)$ is a *convex* function of the speed.

Several solutions for the *on-line* case (only the jobs released at or before time t can be used to compute the speed $s(t)$) have also been investigated in [7]. The authors in [1] prove that Optimal Available (OA), proposed in [7] has a competitive ratio (energy spent by the on-line algorithm over energy of the best off-line algorithm, in the worst case) equal to p^p when the power dissipated at speed s is $Q(s) = s^p$. In CMOS circuits, the value of p is roughly 3, so (OA) may spend 27 times more energy than an optimal schedule in the worst case.

* This work has been partially supported by the LabEx PERSYVAL-Lab.

978-1-7281-9581-0/20/$31.00 ©2020 IEEE

Here, we also consider the on-line case and we aim at minimizing the *expected* energy consumption to complete all jobs before their deadlines. In this case, the problem can be seen as a *constrained optimal control problem*:

> *At time t the decision maker (processor) chooses the speed $s(t)$ based on the current* state *(jobs released before t) and the* expected *future job arrivals.*

The goal of the decision maker is to minimize the expected total energy cost while satisfying all the deadline constraints on the jobs.

We consider two cases, depending on the instants when the decisions are made. The first case is a *discrete time control*: decisions can only be made at the release times of the jobs. The second case is a *continuous time control*: at any time $t \in \mathbb{R}$ the controller chooses the current speed. In principle, the continuous controller is more powerful than the discrete one and should achieve a solution that has a lower energy cost. The goal of this paper is to show that this is not true here: discrete decisions are as good as continuous one. This result has two main consequences: The optimal continuous control can be computed efficiently, and implementation in an actual embedded system becomes simpler.

The paper is organized as follows. The first part introduces the notations and the optimal control problem. The second part presents the optimal speed policy when the decisions are taken at release times of the jobs: At the n-release time, the processor chooses a speed $s(n)$ that is used until the new release time opportunity. It also shows how to compute it and gives its computational cost. The third part deals with the optimal speed policy when the decisions are continuous in time: at any time $t \in [0, T]$, the processor chooses the current speed $s(t)$ and we show that, under mild conditions, continuous speed policies do not improve over the discrete version.

II. Presentation of the problem

A. Jobs, Processor Speeds, and Power

We consider a real-time system with one uni-core processor that executes K real-time, sporadic, independent jobs. Each job i is defined by 3 integers: (r_i, c_i, d_i) where $r_i \in \mathbb{N}$ is the release time (or *arrival time*), $c_i \in \mathbb{N}$ is the size (or *workload*), and

$d_i \in \mathbb{N}$ is the relative deadline. We assume that all jobs have a bounded size C and a bounded deadline Δ:

$$\forall i, \quad c_i \leq C, \qquad d_i \leq \Delta.$$

Without loss of generality, we assume $r_n = n$. This means that a new job arrives at each time slot, this job being of size 0 ($c_n = 0$) if no real work actually arrives at time n. The job stream is given by probability distributions: $(P_n(\cdot,\cdot))$ is the probability of the size and the deadline of a job arriving at any time n: For any $0 \leq \gamma \leq C$, and for any $1 \leq \delta \leq \Delta$,

$$P_n(\gamma, \delta) := \mathbb{P}(c_n = \gamma, d_n = \delta) \tag{1}$$

The CPU processing speed s can vary with time and takes values in a finite set $\mathcal{S}$ of speeds between 0 and s_{max}, $s_{\max}$ being the maximal processor speed. When the processor runs at speed s during an interval of size 1, it executes a total amount of work equal to s. Without loss of generality, we will scale the speeds (as well as the sizes of the jobs), so that all the speeds in $\mathcal{S}$ are integers.

We consider that the power dissipated by the CPU working at speed $s(t)$ at time t is $Q(s(t))$, so that the energy consumption of the processor from time 0 to time T is $J = \int_0^T Q(s(t))dt$. Classical choices for the power Q are convex increasing functions of the speed: in CMOS circuits, we typically have $Q(s) = bs^p$ where p is a constant between 2 and 3.

For the sake of simplicity, we only consider the following simple situation: preemption time is null (the processor can switch from one job to another at any time instantaneously), and speed change time is also null.

B. Problem Statement

The objective is to choose at each time t the speed $s(t)$ and the schedule $R(t)$ (which job to execute at time t among the jobs present at time t) in order to minimize the total energy consumption over the total time horizon, while satisfying all the real-time constraints. Furthermore, the choice must be made on-line, *i.e.*, it can only be based on the characteristics of the past and current jobs.

The information (or history) $\mathcal{H}(t)$ at time t is the set of all the past and current jobs together with the past speed selection:

$$\mathcal{H}(t) = \{(r_i, c_i, d_i) | r_i \leq t\} \cup \{s(u), u \leq t\} \tag{2}$$

Notice that in this model, unlike in [4] or [5], the size c_i and the deadline d_i are known at the release time of job i. The on-line energy minimization problem $(\mathcal{P})$ is:

> *Find online speed $s(t)$ and schedule $R(t)$ (*i.e.*, $s(t)$ and $R(t)$ can only depend on the history $\mathcal{H}(t)$) to minimize the expectancy $\mathbb{E}\int_0^T Q(s(t))dt$ under the constraint that no job misses its deadline.*

Let (s^*, R^*) be an optimal solution to the problem $(\mathcal{P})$. Since the energy consumption does not depend on the schedule (preemption is assumed to be time and energy-free) and since the *Earliest Deadline First*[1] (EDF) scheduling policy is optimal in terms of deadline constraints (see for example [3]), then (s^*, EDF) is also an optimal solution to problem $(\mathcal{P})$. In the following, we will always assume with no loss of optimality that the processor uses EDF to schedule its jobs. This implies that we only need to focus on the speed of the processor to obtain an optimal schedule.

[1] At time t, execute the job whose deadline is the earliest, among all jobs released before t and not yet completed.

III. Discrete Optimization

In this section we consider the case where the processor can only change its speed at the times when jobs (possibly null) arrive in the system. This will make the computation of the optimal speed policy easier but may result *a priori* in a sub-optimal result. This case has been presented in more details with a more general model, including general arrival sequences, context switch costs and delays for changing speeds, in [2]. Here it is mainly useful as a reference point, to which the analysis for the continuous case will be compared.

A. Consecutive Speeds

The set of available speeds $\mathcal{S}$ is *consecutive* when it contains all the integer speeds between 0 and $s_{\max}$: $\mathcal{S} = \{0, 1, \ldots, s_{\max}\}$. When the set $\mathcal{S}$ is not consecutive, one can construct the consecutive set $\overline{\mathcal{S}}$ that contains all the speeds missing in $\mathcal{S}$. We claim that any missing speed $s \in \overline{\mathcal{S}} \setminus \mathcal{S}$ can be emulated by using speeds in $\mathcal{S}$ over a discrete time interval $[n, n+1)$ in the following way:

First, let us define the two *neighboring speeds* s_1 and s_2 of s in $\mathcal{S}$: we have $s_1 < s < s_2$. Let us also define $0 < \alpha < 1$ such that $s = \alpha s_1 + (1-\alpha)s_2$.

Over the interval $[n, n+1)$, if the processor uses speed s_1 over $[n, n+\alpha)$ and speed s_2 over $[n+\alpha, n+1)$, then the total amount of work executed over the whole interval is the same as if speed s were used over the whole interval. The energy cost is $\alpha Q(s_1) + (1-\alpha)Q(s_2)$.

Finally, the behavior of the system is the same as if the speed s were available with a power function $Q(s) = \alpha Q(s_1) + (1-\alpha)Q(s_2)$. This trick (emulating s with two speeds) will be called *V_{dd}-hopping* in the following.

Therefore, in the rest of this section, we always assume with no loss of optimality that the set of speeds is consecutive (by using *V_{dd}-hopping*). We will discuss this issue again in the continuous case.

B. Remaining work function

The remaining work at time n is an increasing function $w_n(\cdot)$ defined as follows: $w_n(u)$ is the amount of work that remains to be done before time $n+u$. Since all available speeds, job sizes, and deadlines are integer numbers, the remaining work $w_n(u)$ is an integer valued staircase function (assumed to be right continuous with left limits).

This is illustrated in Figure 1 that shows an example of the remaining function $w_n(\cdot)$. The jobs released just before $n=4$ are: $(0,2,4)$, $(1,1,5)$, $(2,2,6)$, $(3,2,4)$, and $(4,0,6)$. The last job having a null size, it is not shown in this figure. The speeds chosen by the processor up to time $n=4$ are: $s_0 = 1$, $s_1 = 0$, $s_2 = 2$, and $s_3 = 1$. Function $A(t)$ in green is the amount of work that has arrived before time t. Function $D(t)$ in blue is made of discrete steps that show the cumulative work that must be

executed before time t. This requires a detailed explanation: the first step of $D(1)$ corresponds to the deadline of the first job at $n = 0+4 = 4$; the second step is for the second job at $n = 1+5 = 6$; the third step is for the 4th job at $n = 3+4 = 7$; the fourth step is for 3rd job at $n = 2+6 = 8$. Hence the step for the 4th job is *before* the step for the 3rd. This is because Figure 1 depicts the situation at $t=4$. At $t=3$, we would only have seen the first 3 jobs. Finally, the function $e(t)$ in black is the amount of work already executed by the processor at time t; in Figure 1, the depicted function $e(t)$ has been obtained with an arbitrary speed policy (*i.e.*, non optimal). Finally, the remaining work function $w_n(u)$ in red is exactly the portion of $D(t)$ that remains "above" $e(t)$.

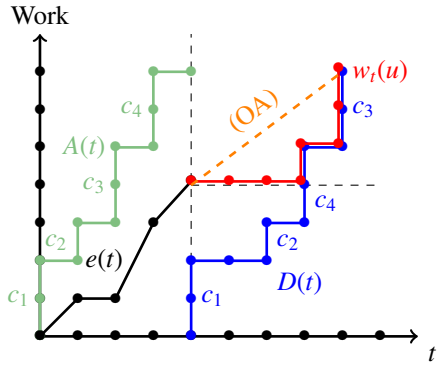

Fig. 1. Construction of the remaining work function $w_n(\cdot)$ at $n = 4$, for 4 jobs $(0, 2, 4), (1, 1, 5), (2, 2, 6), (3, 2, 4), (4, 0, 6)$, and processor speeds $s_0 = 1, s_1 = 0, s_2 = 2, s_3 = 1$. $A(t)$ is the amount of work that has arrived before time t. $D(t)$ is the amount of work that must be executed before time t. $e(t)$ is the amount of work already executed by the processor at time t.

The remaining work function $w_n(\cdot)$ is the only relevant information at time n, out of the whole history $\mathcal{H}(n)$, needed by the processor to choose its next speed. In the following, we call w_n the *state* of the system at time n.

C. Description of the State Space

To formally describe the state space $\mathcal{W}$ (*i.e.*, all the possible remaining work functions) and the evolution of the state w_n over time, we introduce several constructors.

Definition 1. *We define the following operators:*

- *The time shift operator $\mathbb{T}f$ is the shift on the time axis of function f, defined as:* $\forall t \in \mathbb{R}, \quad \mathbb{T}f(t) = f(t+1)$.
- *The positive part of a function f is* $f^+ = max(f, 0)$.
- *The unit step function (Heaviside function), denoted H_t, is the discontinuous step function such that* $\forall u \in \mathbb{R}$:

$$H_t(u) = \{0 \text{ if } u < t \text{ and } 1 \text{ if } u \geq t\}$$

Lemma 1. *Let (r_n, c_n, d_n) be a job that arrives at time $n = r_n$. If the processor speed at time $n-1$ is s_{n-1}, then at time n the remaining work function becomes:*

$$w_n(\cdot) = \mathbb{T}[(w_{n-1}(\cdot) - s_{n-1})^+] + c_n H_{d_n}(\cdot) \tag{3}$$

Proof. Between times $n-1$ and n, the processor working at speed s_{n-1} executes s_{n-1} amount of work, so the remaining work decreases by s_{n-1}. The remaining work cannot be negative by definition, hence the term $(w_{n-1}(\cdot) - s_{n-1})^+$. After a time shift by one unit, a new job i is released at time n, bringing c_n additional work with deadline $n + d_n$, hence the additional term $c_n H_{d_n}(\cdot)$. □

In the following, we will use the notation $w_{n+1}(\cdot) = F(w_n(\cdot), s, a_n)$ to denote the state that follows immediately state w_n under speed s and work arrival a_n at time n.

The state space $\mathcal{W}$ is finite: it is included in the set of all integer staircase functions, with at most Δ steps, each on them of integer height at most C. The following proposition gives a more precise evaluation of the size of the size of the state space $\mathcal{W}$, using a generalization of Catalan numbers. The proof is in [2].

Proposition 1. *If C is the maximal size of a job and Δ its maximal deadline, then the size $G(C, \Delta)$ of the state space $\mathcal{W}$ satisfies:*

$$G(C, \Delta) = \frac{1}{1 + C(\Delta+1)} \binom{(C+1)(\Delta+1)}{\Delta+1} \tag{4}$$

$$\approx \frac{e}{\sqrt{2\pi}} \frac{1}{(\Delta+1)^{3/2}} (e\,C)^{\Delta} \tag{5}$$

D. Markov Decision Process

Since the state space $\mathcal{W}$ is finite in discrete time, one can effectively compute the optimal speeds in each possible state. In this section, we provide a dynamic programming algorithm to compute this optimal speed selection. We compute offline the optimal *speed policy* σ_n^* that gives the speed the processor should use at time n in all its possible states. At runtime, at each time n, the processor chooses the speed that corresponds to its current remaining work w_n as $s^*(n) := \sigma_n^*(w_n)$.

The algorithm to compute the speed policy σ^* is based on a *Markovian evolution* of the jobs.

The work arriving function is denoted by A_n: $A_n(\cdot) = cH_d(\cdot)$, if the job arriving at time n is of size c and relative deadline d.

Therefore, the probability distribution can also be written in terms of arriving functions instead of jobs: with $a(\cdot) = cH_d(\cdot)$,

$$P_n(a) = P_n(c_n = c, d_n = d).$$

Now, we can compute the minimal total expected processor energy consumption in discrete time ($J^{*,\mathbb{N}}$) from 0 to T. We thus want to solve the following optimization problem ($\mathcal{P}_F$). If the initial state is w_0, then compute

$$J^{*,\mathbb{N}}(w_0) = \min_{\sigma} \left\{ \mathbb{E}\left(\sum_{n=0}^{T} Q(\sigma_n(w_n)) \right) \right\} \tag{6}$$

where the minimum is taken over all *policies* σ: for all time n and all state w, $\sigma_n(w)$ denotes the speed used over the time interval $[n, n+1)$ if the current state is w, under speed policy σ.

There are two constraints on σ. Firstly, it must takes its values in the set of admissible speeds, *i.e.*, $\sigma_n(w) \in \mathcal{S}$. Secondly, it must be large enough to execute the remaining work over the next interval $[n, n+1)$, *i.e.* $\sigma_n(w) \geq w(1)$.

This set of *admissible speeds* in state w will be denoted $\mathcal{A}(w)$ and is therefore:

$$\mathcal{A}(w) = \{s \in \mathcal{S} \text{ s.t. } s \geq w(1)\} \tag{7}$$

If the speed policy σ_n chooses a speed in $\mathcal{A}(w)$ for each state w and each time n, then it will never miss any deadline over the whole interval $[0, T]$. Of course, this requires that $\mathcal{A}(w)$ is not empty, or in other words, $w(1) \leq s_{\max}$. This issue will be discussed in Section III-E.

Now, $J^{*,\mathbb{N}}$ can be computed using a *backward induction*. Let $J_n^{*,\mathbb{N}}(w)$ be the minimal expected energy consumption from time n to T, if the state at time n is w ($w_n = w$).

We use Algorithm 1, a backward induction (dynamic programming) to recursively evaluate the expected consumption. Specifically, we use the classical Finite Horizon-Policy Evaluation Algorithm from [6] (p. 80). The *optimal speed policy* returned by Algorithm 1 is the union of all the optimal speeds for each $n \in [0, T-1]$.

Algorithm 1 Dynamic Programming Algorithm (DP) to compute the optimal speed for each state at each time.

$n = T$ % Start at the time horizon T
for all $w \in \mathcal{W}$ **do** $J_T^{*,\mathbb{N}}(w) = 0$ **end for** % Initialize the state at the horizon T
while $n \geq 1$ **do**
  for all $w \in \mathcal{W}$ **do**
    $J_{n-1}^{*,\mathbb{N}}(w) = \min_{s \in \mathcal{A}(w)} \left\{ Q(s) + \sum_a P_n(a) J_n^{*,\mathbb{N}}\Big(F(w,s,a)\Big) \right\}$ %
    Compute the optimal state at $n-1$

    $\sigma_{n-1}^*(w) = \arg\min_{s \in \mathcal{A}(w)} \left\{ Q(s) + \sum_a P_n(a) J_n^{*,\mathbb{N}}\Big(F(w,s,a)\Big) \right\}$
    % And the corresponding optimal speed
  end for
  $n \leftarrow n - 1$ % Move backward
end while
return $\bigcup_{n=0}^{T-1} \sigma_n^*(\cdot)$ % Return the optimal speed policy

The complexity to compute the optimal speed policy $\sigma_n^*(w)$ for all possible states and time steps is finite. It is equal to $O(T|\mathcal{S}|C\Delta G(C,\Delta))$. The combinatorial explosion in maximum deadline, Δ, of the state space makes this complexity very large when Δ is large.

E. Feasibility Issues

A set of jobs is *schedulable* if there exists a speed policy that can execute them all without missing a deadline, regardless of energy costs.

By monotonicity of the remaining work w.r.t. the current speed, it is straightforward to check that using the maximal speed $s_{\max}$ all the time is the best speed policy as far as deadline constraints are concerned. Hence, a set of jobs is schedulable if and only if using the maximal speed all the time will not miss any deadline.

Definition 2 (feasible speed policy)**.** *A speed policy σ is feasible over a set of jobs if using speed $\sigma(\cdot)$, the processor executes all jobs without missing a deadline.*

The goal of this section is to show the following result, which proves that DP is not only optimal in terms of energy but also optimal in terms of feasibility.

Proposition 2. *A finite set of jobs is schedulable if and only if speed policy* DP *is feasible over this set.*

Proof. We will show that for all states w_n reached under DP and all time n, executing a schedulable set of jobs, the set of admissible speeds $\mathcal{A}(w_n)$ is never empty. This is equivalent to the feasibility of DP.

To show this, let us first modify the processor by allowing unbounded speeds, and let us introduce a new energy function $Q'(\cdot)$ such that $\forall s \geq s_{\max}$, $Q'(s) = \infty$. For speed values smaller than $s_{\max}$, the power function remains unchanged: if $s \in \mathcal{S}$, then $Q'(s) = Q(s)$. In this new framework, the processor can now use unbounded speeds:

$$\mathcal{S}' = \mathcal{S} \cup \{s_{\max} + 1, s_{\max} + 2, \ldots\}$$

In that case the new set of admissible speeds

$$\mathcal{A}'(w) = \{s \in \mathcal{S}' \text{ s.t. } s \geq w(1)\}$$

is never empty so that all deadlines are met. However, when DP uses a speed higher than $s_{\max}$, its energy consumption becomes infinite.

Now, let us consider a set of schedulable jobs executed by this extended processor model. Let us also define another speed policy, namely, the simple greedy speed policy, that uses speed $s_{\max}$ at any time n. Under this greedy speed policy, the expected energy consumption is $J^{greedy} = TQ(s_{\max})$ and no job misses its deadline, by schedulability.

Since the optimal speed policy computed by DP is optimal in energy, it has a better expected consumption than the greedy speed policy. Hence, J^* for DP is finite, smaller than $TQ(s_{\max})$. This implies that speeds higher than $s_{\max}$, which would have resulted in an infinite energy consumption, will never be used by the optimal speed policy.

To sum up, for any schedulable set of jobs, the set $\mathcal{A}'(w_n)$ always contains speeds smaller than $s_{\max}$. The optimal speed policy never uses a speed higher than $s_{\max}$ so that it coincides with the original speed policy. Therefore the optimal policy, as defined in Algorithm 1, will never miss a deadline if and only if the set of jobs is schedulable. □

On a side note, since the amount of work that can arrive at any time n is bounded by C, one can assess more explicitly a schedulability condition of a set of jobs. A sufficient schedulability condition is:

$$s_{\max} \geq C. \tag{8}$$

The case where $s_{\max} = C$ is borderline because the only optimal speed policy DP degenerates to a trivial greedy speed policy: at any time n DP executes all the arriving work: $\sigma_n^*(w_n) = A_n(\Delta)$ (recall that $A_n(\Delta)$ is the total work arriving at time n). Since $A_n(\Delta) \leq C \leq s_{\max}$ by (8), this speed policy is feasible.

If $s_{\max} > C$, then the previous speed policy is still feasible but will not be optimal in terms of energy consumption. In

that case the processor has more flexibility and may use a low speed to save energy. Doing so, it leaves some work for later, hoping for non busy times. In the unlikely event that a lot of work arrives in the future, the processor will still be able to use a very large speed to execute it.

F. Monotonicity Properties

Proposition 3. *For any n and any two states $w_n^1(\cdot)$ and $w_n^2(\cdot)$, $w_n^1(\cdot) \geq w_n^2(\cdot)$ implies $\sigma_n^*(w_n^1) \geq \sigma_n^*(w_n^2)$.*

Proof. Consider two systems starting at n, with respective states $w_n^1(\cdot)$ and $w_n^2(\cdot)$. We "couple" the job arrivals $a_n, a_{n+1}, \ldots$ in both systems and use the optimal speeds for the first system, namely:

$$\begin{aligned} s_n &= \sigma_n^*\left(w_n^1\right), \\ s_{n+1} &= \sigma_n^*\left(F(w_n^1, s_n, a_n)\right), \\ s_{n+2} &= \sigma_n^*\left(F(w_{n+1}^1, s_{n+1}, a_{n+1})\right), \\ &\vdots \\ s_{T-1} &= \sigma_n^*\left(F(w_{T-2}^1, s_{T-2}, a_{T-2})\right). \end{aligned}$$

Hence the states in the first system are

$$\begin{aligned} w_{n+1}^1 &= F(w_n^1, s_n, a_n), \\ &\vdots \\ w_{T-1}^1 &= F(w_{T-2}^1, s_{T-2}, a_{T-2}). \end{aligned}$$

And the states in the second system are

$$\begin{aligned} w_{n+1}^2 &= F(w_n^2, s_n, a_n), \\ &\vdots \\ w_{T-1}^2 &= F(w_{T-2}^2, s_{T-2}, a_{T-2}). \end{aligned}$$

By monotonicity of the function F w.r.t. its first coordinate, the initial comparison between the two systems carries on: $w_{n+i}^1 \geq w_{n+i}^2$ for all i. This implies that the sequence of speeds $s_n, s_{n+1}, \ldots$ is also admissible (*i.e.* $s_{n+i} \geq w_{n+i}^1 \geq w_{n+i}^2$) in the second system. This implies that on all trajectories, a lower speed can be used in the second system at time n. A direct backward induction on n now implies that the optimal speed at time n in the second system is smaller that in the first one. □

IV. Continuous Optimization

Let us now consider the continuous version of energy optimization. To ease the analysis of the continuous case, let us first extend the function Q over the continuous speed interval $[0, s_{\max}]$ by affine interpolation between the points $Q(s), s \in \mathcal{S}$:

For any $s \in [0, s_{\max}]$, if s_1 and s_2 are the two neighboring speeds of s in $\mathcal{S}$, we set $Q(s) = \alpha Q(s_1) + (1-\alpha)Q(s_2)$, where α is defined by $s = \alpha s_1 + (1-\alpha)s_2$.

Notice that if Q is convex over the discrete set $\mathcal{S}$, then the interpolation remains convex over $[0, s_{\max}]$. Notice also that this extension can always be done since the processor will only really use integer speeds anyways. It just artificially makes the processor able to use any (virtual) speed s, as a combination of existing speeds (*V_{dd}-hopping*).

In the continuous case, the processor may change its speed at any time $t \in [0, T]$. However, the jobs and the set of speeds remain discrete.

In that case, the state at time t, under any online speed policy π, is still well-defined and satisfies a differential equation:

$$\forall u \geq 0, \quad w_0^\pi(u) = 0$$

$$\forall t > 0 \text{ and } \forall n \in \mathbb{N} \text{ with } n < t \leq n+1,$$

$$w_t^\pi(u) = \left(w_n^\pi(u) - \int_n^t \pi(w_v^\pi)dv\right)^+ + a_t(u). \quad (9)$$

Let us call $J^{*,\mathbb{R}}$ the minimal expected energy consumption over $[0, T]$, and $\pi_t^{*,\mathbb{R}}(\cdot)$ the optimal speed policy, resulting in state $w_t^*(\cdot)$ at time t.

The energy consumed by the optimal speed policy $\pi_t^{*,\mathbb{R}}$, starting in state w_0, is

$$\begin{aligned} J^{*,\mathbb{R}}(w_0) &= \mathbb{E}\int_0^T Q\left(\pi_t^{*,\mathbb{R}}(w_t^*)\right)\mathrm{d}t & (10) \\ &= \mathbb{E}\sum_{n=0}^{T-1}\int_n^{n+1} Q\left(\pi_t^{*,\mathbb{R}}(w_t^*)\right)\mathrm{d}t \\ &\geq \mathbb{E}\sum_{n=0}^{T-1} Q\Big(\int_n^{n+1} \pi_t^{*,\mathbb{R}}(w_t^*)\,\mathrm{d}t\Big) & (11) \\ &= \mathbb{E}\sum_{n=0}^{T-1} Q\left(s^{mean}(n)\right) & (12) \\ &= \mathbb{E}\sum_{n=0}^{T-1} \alpha Q\left(s_1^*(n)\right) + (1-\alpha)Q\Big(s_2^*(n)\Big) & (13) \end{aligned}$$

Let us detail these computations. Eq. (10) is the definition of the optimal speed policy. Eq. (11) comes from Jensen inequality for the convex function Q extended over $[0, s_{\max}]$.

In Eq. (12), $s^{mean} = \int_n^{n+1} \pi_t^{*,\mathbb{R}}(w_t^*)dt$ is the average speed used between n and $n+1$ by the optimal speed policy.

Finally, $s_1^*(n)$ and $s_2^*(n)$ in Eq. (13) are the neighboring speeds in $\mathcal{S}$ of the average speed $s^{mean}(n)$.

Ineq. (12) implies that the speed policy π^{mean} that chooses constant speed $s^{mean}(n)$ during each time interval n to $n+1$, is better than the optimal speed policy.

Furthermore, since deadlines and release times are integers, there is no deadline constraints in $(n, n+1)$. This means that the speed policy π^{mean} is feasible: using the speeds $s_1^*(n)$ and $s_2^*(n)$ over $[n, n+\alpha)$ and $[n+\alpha, n+1]$, resp. does not miss any deadline.

Therefore, the speed policy π^{mean}, which only changes its (virtual) speed s^{mean} at discrete times, is optimal in continuous time.

The difference with the discrete optimal speed policy is that, in the discrete case, the speed used in each interval must be integer-valued, while π^{mean} may use any virtual speed s^{mean} in the continuous interval $[0, s_{\max}]$.

This additional degree of freedom in the continuous case could be beneficial in terms of energy consumption.

Here is a simple example illustrating this fact: If the processor can use speed $s = 0$ over the sub-interval $[n, n+1/3]$

and speed $s' = 1$ over the sub-interval $[n+1/3, n+1]$, then, in total over the interval $[n, n+1]$, this can be seen as a processor using a virtual speed $s^{mean} = 2/3$, with an energy consumption $Q(0)/3 + 2Q(1)/3$. In the discrete optimization problem such a possibility is not available: over $[n, n+1]$, the processor must use either speed 0 or speed 1. This could be detrimental for the expected energy consumption.

In the following, we will show that this is not the case: choosing integer speeds is always optimal.

Theorem 1. *If the set $\mathcal{S}$ is made of consecutive speeds (*i.e. $\mathcal{S} = \{0, 1, 2, \ldots, s_{\max}\}$*) and the power function Q is increasing and convex, then there is no energy gain for the processor to use non-integer speeds: for all initial state w, $J^{*,\mathbb{R}}(w) = J^{*,\mathbb{N}}(w)$.*

Proof. To keep notations simple, we also skip the super-index in $J^{*,\mathbb{N}}$ in the proof up to the last part of the proof.

Let us first consider that the processor can change its speed at times $t \in \mathbb{N}$ as well as at times $t + 1/2$. We will show that this does not bring any energy gain.

The time horizon is T and the minimal energy at time t under state w can be decomposed into two actions (taken at times t and $t + 1/2$):

$$\begin{aligned} J^*_{t+\frac{1}{2}}(w) &= \min_{v \in \mathcal{A}_2(w)} \left(\frac{Q(v)}{2} + \sum_a P_t(a) J^*_{t+1} \left(\mathbb{T}_2 \left(w - \frac{v}{2} \right)^+ + a \right) \right) \\ J^*_t(w) &= \min_{u \in \mathcal{A}_1(w)} \left(\frac{Q(u)}{2} + J^*_{t+\frac{1}{2}} \left(\mathbb{T}_2 \left(w - \frac{u}{2} \right)^+ \right) \right), \end{aligned}$$

where $\mathcal{A}_2(w) = \{s \in \mathcal{S} : s \geq 2w(1)\}$, $\mathcal{A}_1(w) = \mathcal{S}$ and the operator $\mathbb{T}_2(f)$ only shifts the function f by $1/2$: $\mathbb{T}_2(f(x)) = f(x + 1/2)$.

This is a similar dynamic programming equation as used in Algorithm 1, where we take into account the fact that there are no arrival at time $t + 1/2$, and a modified admissibility condition on the speeds: to meet all deadlines, the speed at time $t + 1/2$ must execute all the work with deadline $t + 1$, hence the speed u must be larger than $2w(1)$ while the speed chosen at time t does not have any constraint: indeed, no job can have a deadline at time $t + 1/2$. These two equations show that the new state space should also include the step functions with step sizes in $\mathbb{N}/2$.

By replacing the value of $J^*_{t+1/2}$ in the second equation, one gets J^*_t as a function of J^*_{t+1}:

$$\begin{aligned} J^*_t(w) &= \min_{u \in \mathcal{A}_1(w), v \in \mathcal{A}_2(w)} \left(\frac{Q(u)}{2} + \frac{Q(v)}{2} \right. \\ &+ \left. \sum_a P_t(a) J^*_{t+1} \left(\mathbb{T} \left(w - \frac{u+v}{2} \right)^+ + a \right) \right) \end{aligned}$$

where we have used the distributivity of + over max to get the second line.

This says precisely what was asserted without proof at the beginning: changing speed at half times is equivalent to choosing half speeds at integer times.

The first property that one can get from the last equation is the following: The speeds u, v achieving the min are such that $|u - v| \leq 1$. Indeed, if $|u - v| > 1$, then one can choose $u', v' \in \mathcal{A}_1(w), \mathcal{A}_2(w)$ such that $|u' - v'| \leq 1$ and $u + v = u' + v'$. By convexity of Q, we have $Q(v)/2 + Q(u)/2 \geq Q(v')/2 + Q(u')/2$, implying that the choice u', v' is better than the choice u, v.

With no loss of generality, we will assume in the following that either $u = v$ (in which case we are back to an integer speed) or $v = u + 1$.

A second property is that both speeds u and $v = u + 1$ are admissible in state w: If $u + 1 \in \mathcal{A}_2(T_2(w - u/2)^+)$, then $u + 1 + u \geq 2w(1)$. This implies $u \geq w(1) - 1/2$, so that $u \geq w(1)$ because both u and $w(1)$ are integers (and of course $u + 1 \geq w(1)$).

We are now ready for the proof, which proceeds by backward induction on t. Let us prove the two following properties simultaneously:

(P1) For all w with integer steps sizes, $J^*_t(w)$ is obtained by using integer speeds only.

(P2) For all w, a and all $u \in \mathcal{A}(w)$, then using $v = u + 1$,

$$\begin{aligned} & J^*_t \left(\left(w - \frac{u+v}{2} \right)^+ + a \right) \\ &= \frac{1}{2} \left(J^*_t \left((w-u)^+ + a \right) + J^*_t \left((w-v)^+ + a \right) \right). \end{aligned}$$

Both properties are obviously true at time T where there is nothing to prove. Now, let us prove (P1) at time t:

Under state w with integer steps, let us consider the randomized speed policy that chooses at time t, speed $u \in \mathcal{A}(w)$ with probability $1/2$ and speed $u + 1$ with probability $1/2$ and is optimal from time $t + 1$ on.

The energy of this speed policy is

$$\begin{aligned} & \frac{1}{2} \left(Q(u) + \sum_a P_t(a) J^*_{t+1} \left(\mathbb{T}(w-u)^+ + a \right) \right) \\ & + \frac{1}{2} \left(Q(u+1) + \sum_a P_t(a) J^*_{t+1} \left(\mathbb{T}(w-u-1)^+ + a \right) \right) \\ = \; & \frac{Q(u)}{2} + \frac{Q(u+1)}{2} \\ & + \sum_a P_t(a) J^*_{t+1} \left(\mathbb{T} \left(w - \frac{u+u+1}{2} \right)^+ + a \right) \end{aligned}$$

where (P2) at time $t+1$ is used for states $(\mathbb{T}(w-u)^+ + a, \mathbb{T}(w-u-1)^+ + a, \mathbb{T}(w - \frac{u+u+1}{2})^+ + a)$ to get the second inequality.

This says that this randomized speed policy has the same energy cost as the deterministic speed policy that uses speed $(u + u + 1)/2$ at time t. The theory of MDP says that there exists an optimal speed policy that does not randomize. Here, this implies that there exists an optimal speed policy at time t that uses an integer speed. This is exactly property (P1).

As for property (P2), we first notice that the arrival of jobs a can be included in the state w for simplicity. Therefore, let us consider two states with integer step sizes, $w_2 := (w - u + 1)^+$ and $w_1 := (w-u)^+$ at time t. Using (P1), the optimal speed used in both states are integers. Let us denote by σ_1 the optimal speed used in state w_1.

Since $w_2 \geq w_1$ point-wise, then by monotony of the total energy with respect to the state and by using Proposition 3 and

an induction on t, the optimal speed σ_2 in state w_2 is higher than σ_1: $\sigma_2 \geq \sigma_1$.

We further claim that $\sigma_2 \leq \sigma_1 + 1$. We show this by contradiction: assume that $\sigma_2 = \sigma_1 + k$, with $k \geq 2$. Convexity of the power implies

$$Q(\sigma_1 + k) - Q(\sigma_1 + 1) \geq Q(\sigma_1 + k - 1) - Q(\sigma_1).$$

Since $\sigma_2 = \sigma_1 + k$ is optimal for w_2, we get

$$\begin{aligned} &Q(\sigma_1 + k) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - u + 1 - \sigma_1 - k\right)^+ + a\right) \\ &< Q(\sigma_1 + 1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - u + 1 - \sigma_1 - 1\right)^+ + a\right). \end{aligned}$$

Together with the previous inequality this implies

$$\begin{aligned} &Q(\sigma_1 + k - 1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - u - \sigma_1 - k + 1\right)^+ + a\right) \\ &\qquad < Q(\sigma_1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - u - \sigma_1\right)^+ + a\right). \end{aligned}$$

The first term is the energy cost of using speed $\sigma_1 + k - 1$ in state w_1. The second term is the energy cost of using speed σ_1 in state w_1. This inequality contradicts the optimality of σ_1. Therefore, $\sigma_2 \leq \sigma_1 + 1$.

Now, let us compute the optimal speed σ_3 in the "middle" state $w_3 = (w - \frac{u+u-1}{2})^+$. By monotonicity, $\sigma_1 \leq \sigma_3 \leq \sigma_2$. Therefore, there only exist two possibilities:
If $\sigma_1 = \sigma_2$ then $\sigma_3 = \sigma_1$. In this case,

$$\begin{aligned} & J^*_t(w_3) \\ = \;& Q(\sigma_1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - \frac{2u-1}{2} - \sigma_1\right)^+ + a\right) \\ = \;& Q(\sigma_1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - \frac{2(u+\sigma_1)-1}{2}\right)^+ + a\right) \\ = \;& \frac{1}{2}\left(Q(\sigma_1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}(w - u - \sigma_1)^+ + a\right)\right) \\ & + \frac{1}{2}\left(Q(\sigma_1) + \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}(w + 1 - u - \sigma_1)^+ + a\right)\right) \\ = \;& \frac{1}{2} J^*_t(w_1) + \frac{1}{2} J^*_t(w_2) \end{aligned}$$

where the third equality comes from (P2) at time $t + 1$.

If $\sigma_2 = \sigma_1 + 1$ then using the same reasoning, we can to prove that $\sigma_1 \leq \sigma_2 \leq \sigma_1 + 1$, then the optimal speed in state w_3 is $\sigma_3 = (\sigma_1 + \sigma_2)/2$. In this case,

$$\begin{aligned} & J^*_t(w_3) \\ = \;& \frac{1}{2} Q(\sigma_1) + \frac{1}{2} Q(\sigma_2) \\ + \;& \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}\left(w - \frac{u+u-1}{2} - \frac{\sigma_1+\sigma_1+1}{2}\right)^+ + a\right) \\ = \;& \frac{1}{2} Q(\sigma_1) + \frac{1}{2} Q(\sigma_2) \\ + \;& \sum_a P_t(a) J^*_{t+1}\left(\mathbb{T}(w - u - \sigma_1)^+ + a\right) \\ = \;& \frac{1}{2} J^*_t(w_1) + \frac{1}{2} J^*_t(w_2). \end{aligned}$$

This shows that changing speeds at half times does not help. A straightforward generalization says that changing speeds at times $t \in \mathbb{N}/2^i$ will not help either for any i. By continuity of the total energy with respect to the speed function, this shows that changing speeds at times $t \in \mathbb{R}$ will not help either: $J^{*,\mathcal{S},\mathbb{R}}(w) = J^{*,\mathcal{S},\mathbb{N}}(w)$. This concludes the proof. □

When the available speeds do not form a consecutive set, it is possible that all the optimal speed schedules use virtual speeds over unit intervals that are not available speeds. Here is a simple example. Consider the following degenerated case: A single arrival at time $t = 0$ with probability one, of a job of size 4 with relative deadline 3: $P_0(4, 3) = 1$ and $P_t(0, 1) = 1$ for all $t > 0$ (no arrival after time 0).

If $\mathcal{S} = \{0, 1, 3\}$ (non-consecutive set), then all optimal speed schedules must use speed $s = 3$ during $1/2$ a unit of time before time 3, speed 1 during $5/2$ units of time before time 3, and then speed 0 from time 3 on. Therefore, a mean speed that is not in $\{0, 1, 3\}$ must be used in at least one time unit interval.

As a side note, if the set of available speeds were consecutive, $\overline{\mathcal{S}} = \{0, 1, 2, 3\}$, then all optimal speed schedules would use speed 2 during one time unit, speed 1 during two time units, and speed 0 from time 3 on.

In the following, we show that if $\mathcal{S}$ is not consecutive, it is always possible to go back to the consecutive case.

Theorem 2. *If the set $\mathcal{S}$ is not consecutive, then the optimal speed policy in continuous time can be constructed using integer speed changing instants under an augmented consecutive set of speeds and then using V_{dd}-*hopping.

Proof. Let $\overline{\mathcal{S}}$ be the extended set of speeds to all integer speeds below $s_{\max}$: $\overline{\mathcal{S}} = \{0, 1, 2, \ldots, s_{\max}\}$.

First, we assign to each non available integer speed a power consumption by using a linear interpolation. More precisely, for each $s < s_{\max}$ and $s \notin \mathcal{S}$, let $s_1, s_2 \in \mathcal{S}$ be the two neighboring speeds in $\mathcal{S}$ such that $s_1 < s < s_2$. The speed s can be seen as a convex combination of s_1 and s_2:

$$s = \alpha s_1 + (1 - \alpha) s_2, \text{ with } \alpha = \frac{s_2 - s}{s_2 - s_1}. \tag{14}$$

We define the power consumption of s as:

$$Q(s) = \alpha Q(s_1) + (1 - \alpha) Q(s_2). \tag{15}$$

Once this is done for each non available speed, we can solve the problem over $\overline{\mathcal{S}}$ with integer speed changing instants (the unavailable speeds being seen as, available with the power cost defined in Eq. (15)). According to our notation, the optimal energy when starting in x is $J^{*,\overline{\mathcal{S}},\mathbb{N}}(x)$. The optimal speed policy with integer speed changing instants are denoted $\{s^*(t)\}_{t\in\mathbb{N}} \in \{0, 1, 2, \ldots, s_{\max}\}$.

The following transformation is done at each integer time step $t \in \mathbb{N}$ (this is *V_{dd}-hopping*). In the time interval $[t, t+1)$, if the optimal speed $s^*(t)$ was not originally available ($s^*(t) \notin \mathcal{S}$), then it is replaced by its two neighboring available speeds s_1 and s_2 over sub-intervals $[t, t+\alpha)$ and $[t+\alpha, t+1)$ respectively. Since the deadlines are integers, no job will miss its deadline during the interval $(t, t + 1)$.

This new speed policy only uses speeds in $\mathcal{S}$ but contains speed changes at non-integer times. We denote by $J^{\text{Vdd},\mathcal{S},\mathbb{R}}(w)$ its energy consumption.

Since the power cost $P_{ower}(s^*(t))$ is a linear interpolation of the power cost of the neighboring available speeds s_1 and s_2, the energy consumption over the interval $[t, t+1)$ is the same using speed $s^*(t)$ on $[t, t+1]$ and using the neighboring speeds s_1 and s_2 over the two sub-intervals $[t, t+\beta]$ and $[t+\beta, t+1]$. This also means that the total energy consumption is the same before and after using *V_{dd}-hopping*:

$$J^{\text{Vdd},\mathcal{S},\mathbb{R}}(w) = J^{*,\overline{\mathcal{S}},\mathbb{N}}(w).$$

On the one hand, Theorem 1 states that, with consecutive speeds, integer speed changing instants minimize the total energy consumption. In other words, this can be written as:

$$J^{*,\overline{\mathcal{S}},\mathbb{R}}(w) = J^{*,\overline{\mathcal{S}},\mathbb{N}}(w).$$

On the other hand, the optimal solution only using the subset composed by the available speeds must use at least as much energy as when all the intermediate speeds are available. This implies

$$J^{*,\overline{\mathcal{S}},\mathbb{R}}(w) \leq J^{*,\mathcal{S},\mathbb{R}}(w).$$

Putting everything together yields the following sequence of inequalities:

$$J^{\text{Vdd},\mathcal{S},\mathbb{R}}(w) \geq J^{*,\mathcal{S},\mathbb{R}}(w) \geq J^{*,\overline{\mathcal{S}},\mathbb{R}}(w) = J^{*,\overline{\mathcal{S}},\mathbb{N}}(w) = J^{\text{Vdd},\mathcal{S},\mathbb{R}}(w).$$

This shows that $J^{\text{Vdd},\mathcal{S},\mathbb{R}}(w) = J^{*,\mathcal{S},\mathbb{R}}(w)$, meaning that the *$V_{dd}$-hopping* speed policy is optimal. This concludes the proof. □

This optimal speed policy can be computed easily: Use Algorithm 1 to compute the optimal solution with integer decision time over the extended consecutive set of speeds and add *V_{dd}-hopping* in all unit intervals where it is necessary.

V. Convexification of the power function

The previous section uses the fact that Q is convex.

However, in most real processors, measurements of the power function show that it is not a convex function of the speed. In most cases, a more realistic approximation is $Q(s) = Q_{stat} + Q_{dyn}(s)$, where $Q_{dyn}(s)$ is convex. An even more accurate model is $Q(s) = Q_{stat}(s) + Q_{dyn}(s)$, where $Q_{dyn}(s)$ is convex but the leakage power $Q_{stat}(s)$ depends on s and is not convex.

If the power function is not convex, then we will show that replacing $Q(\cdot)$ by its convex hull $\widehat{Q}(\cdot)$, and solving the speed selection problem with $\widehat{Q}(\cdot)$ instead of $Q(\cdot)$ also provides the optimal solution with $Q(\cdot)$, by using speed replacements as described in the proof of Theorem 2.

Now we will present how to convexify the Q function. Let us consider a processor, whose speeds belong to the set $\mathcal{S} = \{s_0, s_1, s_2, s_{\max}\}$ and the power function of the processor $Q(\cdot) : \mathcal{S} \to \mathbb{R}$.

If the power function is not convex, some speeds are not relevant, because using these speeds is more expensive in term of energy than using a combination of other speeds. It is always better to only select the speeds whose power consumption belongs to the convex hull of the power function. Indeed if $\widehat{Q}(s) < Q(s)$, then, instead of selecting speed s during any time interval $[t, t+1)$, the processor can select speed s_1 during a fraction of time α_1, and then speed s_2 during a fraction of time α_2, such that $\alpha_1 s_1 + \alpha_2 s_2 = s$, and such that $\widehat{Q}(s) = \alpha_1 Q(s_1) + \alpha_2 Q(s_2)$. The total quantity of work executed during the time interval $[t, t+1)$ will be the same as with s, but the energy consumption will instead be $\widehat{Q}(s) = \alpha_1 Q(s_1) + \alpha_2 Q(s_2)$, which is less than $Q(s)$. Again, this approach uses the *V_{dd}-hopping* technique.

As a result, we can always consider that the power function is convex, as long as switching from one speed to another is instantaneous and has no energetic cost. This is very useful and rather realistic in practice. Indeed, the actual power consumption of a CMOS circuit working at speed s is a non-convex function of the form $Q(s) = bs^p + L(s)$, where the constant b depends on the activation of the logical gates, p is between 2 and 3, and $L(s)$ is the leakage, with $L(0) = 0$ and $L(s) \neq 0$ if $s > 0$. In this case, convexification removes the small values of s from the set of useful speeds.

VI. Conclusion

We have studied the problem of continuous time minimization of the energy consumption of a processor with a discrete set of speeds executing jobs with discrete features (arrival times, sizes and deadlines). Our main result is that this *continuous time* optimization problem can be solved in *discrete time*. This is very useful in practice because the discrete optimal speed policy can be computed effectively using a finite (admittedly large) Markov Decision Process. Moreover, a discrete time speed policy is much easier to implement in an embedded system.

References

[1] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1), 2007.

[2] Bruno Gaujal, Alain Girault, and Stephan Plassart. Dynamic speed scaling minimizing expected energy consumption for real-time tasks. Technical Report hal-01615835, Inria, 2017.

[3] Bruno Gaujal, Alain Girault, and Stéphan Plassart. A Discrete Time Markov Decision Process for Energy Minimization Under Deadline Constraints. Research report, Grenoble Alpes ; Inria Grenoble Rhône-Alpes, Université de Grenoble, December 2019.

[4] F. Gruian. On energy reduction in hard real-time systems containing tasks with stochastic execution times. In *IEEE Workshop on Power Management for Real-Time and Embedded Systems*, pages 11–16, 2001.

[5] J.R. Lorch and A.J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *ACM SIGMETRICS 2001 Conference*, pages 50–61, 2001.

[6] Martin L. Puterman. *Markov Decision Process : Discrete Stochastic Dynamic Programming*. Wiley, wiley series in probability and statistics edition, February 2005.

[7] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of IEEE Annual Foundations of Computer Science*, pages 374–382, 1995.